



Leseprobe

Dirk Louis, Peter Müller

Android

Der schnelle und einfache Einstieg in die Programmierung und  
Entwicklungsumgebung

ISBN (Buch): 978-3-446-44598-7

ISBN (E-Book): 978-3-446-45112-4

Weitere Informationen oder Bestellungen unter

<http://www.hanser-fachbuch.de/978-3-446-44598-7>

sowie im Buchhandel.

# Inhalt

<b>Vorwort</b> .....	<b>XV</b>
<b>Teil I: Einführung</b> .....	<b>1</b>
<b>1 Der Rechner wird vorbereitet</b> .....	<b>3</b>
1.1 Die nötigen Hilfsmittel .....	3
1.2 Installation des JDK .....	4
1.3 Installation von Android Studio .....	5
1.3.1 Erster Start .....	7
1.4 Der Android-SDK-Manager .....	9
1.4.1 Die Android-Versionen .....	9
1.4.2 APIs/SDKs und anderes nachinstallieren .....	10
1.4.3 Dokumentation und API-Referenz .....	12
1.5 Wo Sie weitere Hilfe finden .....	14
1.6 Nächste Schritte .....	15
1.7 Frage und Antworten .....	15
1.8 Übungen .....	15
<b>2 Auf die Plätze, fertig ... App!</b> .....	<b>17</b>
2.1 Die Ruhe vor dem Sturm .....	17
2.2 Das Projekt .....	18
2.3 Das vorgegebene Codegerüst .....	26
2.3.1 Die package-Anweisung .....	28
2.3.2 Die import-Anweisungen .....	28
2.3.3 Die Klassendefinition .....	29
2.4 Layout und Ressourcen .....	31
2.4.1 XML-Layouts .....	31
2.4.2 Ressourcen .....	34
2.5 Die App bauen (Build) .....	37

2.6	Die App im Emulator testen .....	38
2.6.1	AVD für Emulator anlegen .....	38
2.6.2	Die App testen .....	41
2.7	Die App auf Smartphone oder Tablet testen .....	45
2.8	Nächste Schritte .....	47
2.9	Fragen und Antworten .....	47
2.10	Übungen .....	48
<b>3</b>	<b>Was wann wofür .....</b>	<b>49</b>
3.1	Was ist zu tun? – Die drei Pfeiler der App-Erstellung .....	49
3.2	Wer hilft uns? – Bausteine und Klassen .....	50
3.2.1	Bausteine für den App-Aufbau .....	50
3.2.2	Klassen zur Adressierung spezieller Aufgaben .....	54
3.3	Wo wird was gespeichert? – Dateitypen, die Sie kennen sollten .....	55
3.3.1	Quelldateien .....	56
3.3.2	Die Datei R.java .....	56
3.3.3	Assets .....	57
3.3.4	Die Ressourcendateien .....	57
3.3.5	Die Manifestdatei <i>AndroidManifest.xml</i> .....	58
3.3.6	Die APK-Datei .....	59
3.4	Frage und Antworten .....	59
3.5	Übung .....	60
	<b>Teil II: Grundlagen .....</b>	<b>61</b>
<b>4</b>	<b>Code .....</b>	<b>63</b>
4.1	Der Editor .....	63
4.1.1	Syntaxhervorhebung .....	64
4.1.2	Code Folding (Code-Gliederung) .....	65
4.1.3	Code Completion (Code-Vervollständigung) .....	67
4.1.4	Syntaxfehler beheben .....	69
4.1.5	Informationen über Klassen und Methoden .....	71
4.1.6	Klammerpaare identifizieren .....	74
4.1.7	Zeilennummern einblenden .....	74
4.1.8	Code generieren .....	75
4.1.9	Refactoring (Code umstrukturieren) .....	78
4.1.10	Dateiverlauf (Local History) .....	81
4.2	Neue Klassen anlegen .....	81
4.3	Fragen und Antworten .....	84
4.4	Übungen .....	84

<b>5</b>	<b>Benutzeroberfläche und Layout</b>	<b>85</b>
5.1	Einführung	85
5.2	Der Layout-Designer	86
5.2.1	Die Text-Ansicht (XML-Code)	88
5.2.2	Die Design-Ansicht	92
5.3	Layouts (ViewGroups)	95
5.3.1	Die allgemeinen Layoutparameter	96
5.3.2	ViewGroups	99
5.3.3	Hintergrundfarbe (oder -bild)	109
5.3.4	Der Hierarchy Viewer	112
5.4	UI-Elemente	114
5.5	Richtlinien für das Design von Benutzeroberflächen	118
5.6	Praxisbeispiel: eine Quiz-Oberfläche	121
5.7	Hoch- und Querformat	126
5.8	Das App-Symbol	127
5.9	Views im Code verwenden	128
5.9.1	Layouts laden	128
5.9.2	Zugriff auf UI-Elemente	129
5.10	Fragen und Antworten	131
5.11	Übung	131
<b>6</b>	<b>Ressourcen</b>	<b>133</b>
6.1	Der grundlegende Umgang	133
6.1.1	Ressourcen anlegen	134
6.1.2	Ressourcen verwenden	136
6.1.3	Ressourcen aus dem Projekt entfernen	140
6.2	Welche Arten von Ressourcen gibt es?	140
6.2.1	Größenangaben	140
6.2.2	Farben	141
6.2.3	Strings	142
6.2.4	Strings in mehreren Sprachen (Lokalisierung)	144
6.2.5	Bilder	145
6.2.6	Layouts	146
6.2.7	Menüs	147
6.2.8	Roh- und Multimediadaten	147
6.2.9	Stile	148
6.3	Alternative Ressourcen vorsehen	152
6.3.1	Das Grundprinzip	152
6.3.2	Wie stellt man konfigurationsspezifische Ressourcen bereit?	153
6.4	Fragen und Antworten	155
6.5	Übungen	156

<b>7</b>	<b>Mit dem Anwender interagieren</b>	<b>157</b>
7.1	Das Grundprinzip	157
7.1.1	Auf ein Ereignis reagieren	158
7.1.2	Welche Ereignisse gibt es?	161
7.1.3	Hintergrund der Ereignisverarbeitung	162
7.2	Vereinfachte Ereignisbehandlung	164
7.2.1	Ereignisbehandlung mit anonymen Listener-Klassen	165
7.2.2	Ereignisbehandlung mit anonymen Listener-Objekten	166
7.2.3	Ereignisbehandlung mithilfe der Activity-Klasse	166
7.3	Eine Behandlungsmethode für mehrere Views	167
7.4	Auf Tipp- und Wischereignisse reagieren	168
7.4.1	Tippereignisse	168
7.4.2	Wischereignisse	170
7.5	Multi-Touch und Gesten erkennen	172
7.5.1	Multi-Touch	172
7.5.2	Gestenerkennung	174
7.6	Frage und Antworten	176
7.7	Übung	176
<b>8</b>	<b>App-Grundlagen und Lebenszyklus</b>	<b>177</b>
8.1	Die Android-Architektur	177
8.2	Der App-Lebenszyklus	179
8.3	Der Activity-Lebenszyklus	181
8.4	Lebenszyklusdemo	183
8.5	Fragen und Antworten	187
8.6	Übung	187
<b>Teil III: Weiterführende Themen</b>		<b>189</b>
<b>9</b>	<b>In Views zeichnen</b>	<b>191</b>
9.1	Das Grundprinzip	191
9.1.1	Die Leinwand	191
9.1.2	Das Atelier	191
9.1.3	Die Zeichenmethoden und -werkzeuge	192
9.1.4	Wie alles zusammenwirkt	192
9.2	Grafikprimitive zeichnen	196
9.3	Bilder laden	200
9.4	In Bilder hineinzeichnen	201
9.5	Bilder bewegen	203
9.6	Verbesserungen	208
9.7	Fragen und Antworten	209
9.8	Übung	209

<b>10</b>	<b>Menüs, Fragmente und Dialoge</b>	<b>211</b>
10.1	Menüs	211
10.1.1	Menüverwirrungen	212
10.1.2	Menüressourcen	213
10.1.3	Menüeinträge in der ActionBar (AppBar)	215
10.1.4	Das Optionen-Menü	216
10.1.5	Das Kontextmenü	217
10.1.6	Popup-Menü	219
10.1.7	Untermenüs	220
10.1.8	Auf die Auswahl eines Menüeintrags reagieren	221
10.2	Fragmente	223
10.2.1	Was ist ein Fragment?	223
10.2.2	Ein Fragment erzeugen	224
10.2.3	Fragment zur Activity hinzufügen	225
10.2.4	Ein Fragmentbeispiel	226
10.2.5	Definition der Fragment-Klassen	229
10.2.6	Die Activity	231
10.3	Dialoge	233
10.3.1	Dialoge erzeugen	234
10.3.2	Dialoge anzeigen	235
10.3.3	Standarddialoge mit AlertDialog	235
10.3.4	Dialoge für Datums- und Zeitauswahl	237
10.3.5	Der Fortschrittsdialog	240
10.3.6	Eigene Dialoge definieren	242
10.4	Benachrichtigungen mit Toasts	244
10.4.1	Toasts im Hintergrund-Thread	244
10.5	Fragen und Antworten	245
10.6	Übungen	246
<b>11</b>	<b>Mehrseitige Apps</b>	<b>247</b>
11.1	Intents	247
11.1.1	Was sind Intents?	248
11.1.2	Explizite und implizite Intents	249
11.1.3	Intent-Filter	249
11.2	Activities starten mit Intents	250
11.2.1	Intent-Objekte erzeugen	251
11.3	Intents empfangen	253
11.4	Ein Demo-Beispiel	253
11.5	Ergebnisse zurücksenden	256
11.6	Fragen und Antworten	257
11.7	Übung	257

<b>12</b>	<b>Daten speichern</b>	<b>259</b>
12.1	Preferences	259
12.2	Dateizugriffe	260
12.2.1	Zugriff auf internen Speicher	261
12.2.2	Externer Speicher (SD-Karte)	264
12.3	Die Reaktions-App	267
12.4	Fragen und Antworten	272
12.5	Übungen	272
<b>13</b>	<b>Quiz-Apps</b>	<b>273</b>
13.1	Aufbau und Benutzeroberfläche	273
13.2	Die Activity (QuizActivity.java)	274
13.3	Die Fragen (Frage.java)	276
13.4	Die Spielsteuerung (SpielLogik.java)	277
13.5	Verbesserungen	279
13.6	Frage und Antwort	280
13.7	Übung	280
<b>14</b>	<b>Multimedia</b>	<b>281</b>
14.1	Audioressourcen	281
14.2	Soundeffekte mit SoundPool	282
14.3	Das Universalgenie: MediaPlayer	284
14.3.1	Audioressourcen abspielen	284
14.3.2	Audiodateien vom Dateisystem abspielen	285
14.3.3	Audiodateien aus dem Internet abspielen	285
14.3.4	Auf das Abspielende reagieren	286
14.3.5	MediaPlayer-Objekte wiederverwenden	287
14.3.6	Ressourcen freigeben	289
14.3.7	Audiodateien wiederholt abspielen	290
14.4	Piepen und andere Töne	290
14.5	Bilddateien anzeigen	292
14.6	Videos abspielen	293
14.7	Fotos und Videos aufnehmen	295
14.8	Fragen und Antworten	298
14.9	Übungen	298
<b>15</b>	<b>Sensoren</b>	<b>299</b>
15.1	Zugriff	299
15.1.1	Was Sie benötigen	300
15.1.2	Welche Sensoren sind verfügbar?	300
15.1.3	Anmeldung beim Sensor	302

15.2	Sensordaten auslesen .....	303
15.2.1	Beschleunigungswerte ermitteln .....	305
15.2.2	Lagedaten ermitteln .....	309
15.3	Fragen und Antworten .....	312
15.4	Übung .....	313
<b>16</b>	<b>Einsatz der Datenbank SQLite .....</b>	<b>315</b>
16.1	Was ist eine relationale Datenbank? .....	315
16.2	Datenbank anlegen/öffnen .....	316
16.2.1	onCreate() .....	317
16.2.2	onUpgrade() .....	319
16.2.3	close() .....	319
16.2.4	Datenbanken als Ressourcen mitgeben .....	319
16.3	Datenzugriffe .....	320
16.4	Datenbankinhalte mit ListView anzeigen .....	325
16.5	Fragen und Antworten .....	329
16.6	Übung .....	330
<b>17</b>	<b>Geolokation .....</b>	<b>331</b>
17.1	Zugriff .....	331
17.1.1	Verfügbarkeit feststellen .....	331
17.1.2	Daten empfangen .....	332
17.1.3	Empfänger abmelden .....	333
17.2	Geokoordinaten .....	334
17.2.1	Sexagesimale und dezimale Darstellung .....	334
17.2.2	Das Location-Objekt .....	334
17.3	Eine GPS-Tracker-App .....	336
17.4	Fragen und Antworten .....	340
17.5	Übung .....	340
<b>18</b>	<b>Brettspiel-Apps (TicTacToe) .....</b>	<b>341</b>
18.1	Aufbau und Benutzeroberfläche .....	341
18.2	Die Start-Activity (MainActivity) .....	343
18.3	Spielfeld und Logik (TicTacToeView) .....	344
18.3.1	Vorbereitungen .....	344
18.3.2	Spielfeld zeichnen .....	345
18.3.3	Spielerzug durchführen .....	347
18.3.4	Computerzug mit AsyncTask durchführen .....	348
18.4	Verbesserungen .....	351
18.5	Frage und Antwort .....	351
18.6	Übung .....	352



<b>19</b>	<b>Tipps und Tricks</b>	<b>353</b>
19.1	Das Smartphone vibrieren lassen	353
19.2	UI-Code periodisch ausführen lassen	354
19.3	Bildergalerien mit GridView und BaseAdapter	357
19.3.1	Die Bildressourcen	358
19.3.2	Die Adapter-Klasse	358
19.3.3	Die GridView	361
19.3.4	Angeklickte Bilder als Vollbild anzeigen	362
19.4	Spinner verwenden (Listenfelder)	364
19.4.1	Den Spinner mit Daten füllen	365
19.4.2	Ereignisbehandlung	366
19.5	Mehrsprachige Apps	367
19.6	Schlussbemerkung	370
	<b>Teil IV: Anhänge</b>	<b>371</b>
	<b>Anhang A: Apps veröffentlichen oder weitergeben</b>	<b>373</b>
A.1	Die App vorbereiten	373
A.2	Digitales Signieren	375
A.3	Die App exportieren und signieren	376
A.4	Bei Google Play registrieren	378
A.4.1	Steuerliche Aspekte bei App-Verkauf	379
A.5	App hochladen	380
A.6	Weitergabe an Bekannte	380
	<b>Anhang B: Android Studio</b>	<b>383</b>
B.1	Android-Projekt anlegen	383
B.2	Projekt bauen (Build)	383
B.3	Projekte löschen	385
B.4	Eclipse-ADT-Projekt importieren	385
B.5	Run-Konfigurationen anpassen	385
B.6	Fenster zurücksetzen	386
B.7	Apps exportieren	386
B.8	Kleines Android Studio-Wörterbuch	386
	<b>Anhang C: Emulator, ADM &amp; Debugger</b>	<b>389</b>
C.1	Der Emulator	389
C.1.1	AVD-Dateien	390
C.1.2	Emulator starten	393
C.1.3	Die Emulator-Bedienung	395
C.1.4	Apps installieren und deinstallieren	396

C.2	Android Device Monitor (ADM) .....	396
C.3	Der Debugger .....	401
C.3.1	Debug-Lauf starten .....	401
C.3.2	Debug-Möglichkeiten .....	402
C.4	Debugging-Beispiel .....	404
<b>Anhang D: Das Material zum Buch .....</b>		<b>409</b>
<b>Anhang E: Lösungen .....</b>		<b>411</b>
<b>Anhang F: Glossar .....</b>		<b>427</b>
<b>Index .....</b>		<b>437</b>

# Vorwort

Willkommen in der Android-Welt! Seitdem sich der Touchscreen als Standardoberfläche von Mobilfunktelefonen etabliert hat und vor Kurzem noch völlig unbekannte Features wie GPS-Empfänger und Lagesensor zur Standardausstattung gehören, gibt es kein Halten mehr: Jede Woche erscheinen neue Android-basierte Geräte und die Zahl der verfügbaren Apps im Android Market explodiert geradezu.

Wenn auch Sie dazugehören wollen, wenn Sie nicht bloß Anwender sein möchten, sondern daran interessiert sind, eigene Ideen in Apps umzusetzen – sei es zum Spaß oder auch vielleicht als Einstieg in eine Existenz als selbstständiger Software-Entwickler –, dann kann Ihnen dieses Buch einen guten Einstieg (und ein bisschen mehr) in die Welt der App-Programmierung für Android-Systeme bieten.

## Vorkenntnisse und Anforderungen

Wir wollen nichts beschönigen. Die Anforderungen an Android-Programmierer sind hoch. Doch mithilfe dieses Buchs und ein wenig Ausdauer und Mitdenken sollten Sie die größten Hürden meistern können.

Sehen wir uns dazu einmal an, welche Fähigkeiten ein Android-Programmierer besitzen muss und inwieweit Ihnen dieses Buch helfen kann, diese Fähigkeiten zu entwickeln.

- *Gute Kenntnisse der Programmiersprache Java*  
Sie erfüllen diesen Punkt nicht? Kein Grund zur Panik, aber lesen Sie unbedingt den nachfolgenden Abschnitt zum „idealen Leser“.
- *Umgang mit der integrierten Entwicklungsumgebung (IDE) Android Studio.*  
Alles, was Sie zum Umgang mit Android Studio im Allgemeinen wie auch im Hinblick auf die Erstellung von Android-Apps wissen müssen, lernen Sie in diesem Buch. Zusätzlich finden Sie am Ende des Buchs einen eigenen Anhang zu Android Studio, wo die wichtigsten Aufgaben noch einmal zusammengefasst sind (inklusive eines kleinen Wörterbuchs, das Lesern, die im Englischen nicht so versiert sind, die Eingewöhnung in die durchweg englische Benutzeroberfläche erleichtern soll).
- *Einsatz verschiedener Hilfsprogramme wie HierarchyViewer, Debugger und Emulator.*  
Insbesondere der Emulator ist für die Entwicklung von Apps unerlässlich, da Sie mit seiner Hilfe unterschiedlich ausgestattete Android-Geräte simulieren („emulieren“) können, ohne sie tatsächlich als echtes Gerät zu besitzen.

Unnötig zu erwähnen, dass wir Ihnen die wichtigsten Hilfsprogramme in diesem Buch vorstellen und Sie in die Arbeit mit ihnen einführen.

- *Wissen um den Aufbau von Apps und Kenntnis der Android-Klassenbibliothek*

Dies ist das eigentliche Thema dieses Buchs. Wie sieht das Grundgerüst einer Android-App aus, worauf muss ich achten und was für tolle Sachen kann man mit der Android-Klassenbibliothek machen? (Kurzantwort: Nichts ist unmöglich!)

Nach dem erfolgreichen Durcharbeiten dieses Buchs werden Sie sicher noch kein Profi-Android-Entwickler sein. Das können und wollen wir Ihnen gar nicht versprechen, denn der Umfang an Material wäre so groß, dass kein Platz mehr für ausführliche Erläuterungen bliebe.

Sie werden aber eine sehr fundierte Grundlage erhalten, in viele fortgeschrittene Bereiche blicken und alles Notwendige lernen, um tolle Apps erstellen und sich selbstständig weiterbilden zu können.

### **Der ideale Leser, Java-Kenntnisse und das Java-Tutorium**

Da es den idealen Leser im Grunde gar nicht gibt, sollten wir uns lieber fragen, welche Lesergruppen in welchem Umfang von dem vorliegenden Buch profitieren können:

Leser mit guten Java-Kenntnissen, die sicher objektorientiert programmieren können und bereits Erfahrung mit Konzepten wie Überschreibung, Interface-Implementierung, Ereignis-Listener und Threads haben, bilden eine der drei Hauptzielgruppen, für die dieses Buch geschrieben wurde. Sollten Sie zu dieser Gruppe zählen, legen Sie einfach los.

Leser mit grundlegenden Java-Kenntnissen bilden die zweite Hauptzielgruppe und sollten mit diesem Buch ebenfalls gut und schnell vorankommen. Sollten Sie zu dieser Gruppe gehören, achten Sie auf die im Buchtext eingestreuten Hinweise zu den Exkursen des Java-Tutoriums unter <http://files.hanser.de/fachbuch/PDFs.zip>. Mithilfe dieser Exkurse können Sie etwaige Wissenslücken zur Java-Programmierung schließen.

Umsteiger von anderen Programmiersprachen bilden die dritte Hauptzielgruppe. Doch Obacht! Es liegt viel Arbeit vor Ihnen, denn Sie müssen sich parallel auch noch mithilfe des Java-Tutoriums in Java einarbeiten. Sofern Sie allerdings bereits über gute Programmierkenntnisse in einer anderen objektorientierten Sprache (wie z. B. C++ oder C#) verfügen, dürfte dies für Sie keine große Schwierigkeit sein. Sie können das Tutorium vorab oder parallel zu diesem Buch lesen (die ersten Kapitel enthalten zu diesem Zweck Hinweise, wann Sie welche Teile des Tutoriums lesen sollten).

Bleibt die Gruppe der Leser, die über keine oder nur wenig Programmiererfahrung verfügen. Angehörigen dieser Gruppe können wir eigentlich nur empfehlen, sich zuerst einmal in die Java-Programmierung einzuarbeiten (beispielsweise mit unserem Java-Titel). Sie können es aber natürlich auch mit dem Java-Tutorium versuchen. Es geht zwar relativ flott voran, ist aber recht gut verständlich und beinhaltet sogar eine allgemeine Einführung in die grundlegenden Programmierkonzepte.

## Aufbau des Buchs

Das Buch ist in drei Teile plus Anhang gegliedert.

- Der erste Teil behandelt die Installation der notwendigen Entwicklerwerkzeuge und die Grundlagen der App-Erstellung.
- Der zweite Teil vertieft die im ersten Teil angesprochenen Grundthemen: Code, Benutzeroberfläche, Arbeiten mit Ressourcen und der App-Lebenszyklus.
- Der dritte Teil behandelt zahlreiche fortgeschrittene Aspekte wie z. B. Grafik, Menüs, Sensoren, Spiele, Datenbanken oder Geolokation. Er unterscheidet sich nicht nur inhaltlich, sondern auch konzeptionell von den beiden vorangehenden Teilen und ist eher im Stile eines Fortgeschrittenenbuchs geschrieben.

Abgerundet wird das Buch mit Anhängen zur Veröffentlichung von Apps, zur Entwicklungsumgebung Android Studio sowie zu weiteren Werkzeugen wie Emulator, ADM und Debugger, einem Glossar und einem ausführlichen Index.

## Software, Beispiel-Apps und sonstiges Material zum Buch

Die Android-Entwicklungsumgebung, die Beispielsammlung und die Tutorials stehen für Sie zum Download bereit. Die Download-Links finden Sie in Anhang D: Das Material zum Buch. Bitte beachten Sie, dass es für die Android-Entwicklungsumgebung und den Java-SDK mehrere Download-Links gibt. Wählen Sie einfach den Link, der zu Ihrem Betriebssystem passt.

## Die Website zum Buch

Wir haben dieses Buch mit großer Sorgfalt erstellt. Falls Sie auf Probleme oder Fehler stoßen, sollten Sie nicht zögern, uns eine E-Mail unter Angabe von Buchtitel und Auflage zu senden. Schauen Sie auch einmal auf unserer Buch-Website

*[www.carpelibrum.de](http://www.carpelibrum.de)*

nach. Neben zusätzlichem Material, den Lösungsprojekten zu den Übungen, Aktualisierungen und Errata finden Sie dort auch weitere Bücher zum Thema Programmieren in Java, C++, C# u. a.

Viel Spaß in der Android-Welt wünschen Ihnen

*Dirk Louis (autoren@carpelibrum.de)*

*Peter Müller (leserfragen@gmx.de)*

# 9

## In Views zeichnen

Der Einsatz von UI-Elementen ist eine Möglichkeit, eine App-Oberfläche zu gestalten. Die andere Möglichkeit ist das direkte Zeichnen in die App-Oberfläche oder gezeichnete Figuren (Sprites) vor einem Hintergrund zu bewegen.

### ■ 9.1 Das Grundprinzip

Um in eine App zeichnen zu können, brauchen wir

- eine *Leinwand* – sprich ein Objekt, in das wir zeichnen können,
- ein *Atelier* – sprich einen Ort, wo wir zeichnen können,
- *Zeichenwerkzeuge* – also Pinsel und Farben.

#### 9.1.1 Die Leinwand

Leinwände sind in der Android-Programmierung Instanzen der Klasse `Canvas`. Jede `View` verfügt über eine solche `Canvas`-Leinwand, weswegen wir grundsätzlich auch jede `View` zum Zeichnen verwenden können. Ideal aber sind natürlich `Views`, die uns eine unverbrauchte, freie Zeichenfläche bieten, wie z.B. die Basisklasse `View` selbst, `ImageView` oder `SurfaceView`.

#### 9.1.2 Das Atelier

Jede `View` verfügt über eine Methode `onDraw()`, die automatisch vom Android-System aufgerufen wird, wenn die `View` sichtbar wird und sich selbst zeichnen soll.

Grundsätzlich sollten Sie Ihren Zeichencode immer in diese Methode packen. Erstens müssen Sie sich dann nicht selbst darum kümmern, dass Sie Zugriff auf das `Canvas`-Objekt der `View` bekommen (dieses wird Ihnen über den Parameter der `onDraw()`-Methode zur Verfü-

gung gestellt). Zweitens ist auf diese Weise sichergestellt, dass Ihr Zeichencode wenn nötig automatisch ausgeführt wird.

Um Zeichencode in die `onDraw()`-Methode einfügen zu können, müssen Sie die Methode allerdings überschreiben – was wiederum bedeutet, dass Sie eine eigene View-Klasse ableiten müssen (vorzugsweise von `View`, `ImageView` oder `SurfaceView`).



Mögliche Alternativen sind das Zeichnen mit `Drawable`-Objekten (für einfache Zeichenoperationen) oder das Zeichnen in eine `SurfaceView` bei gleichzeitiger Implementierung des Interface `SurfaceHolder.Callback`. Auf diese beiden Alternativen werden wir hier allerdings nicht weiter eingehen.

### 9.1.3 Die Zeichenmethoden und -werkzeuge

Das `Canvas`-Objekt repräsentiert für uns nicht nur die Leinwand, in die wir zeichnen, es stellt uns auch gleich die Methoden zur Verfügung, die wir zum Zeichnen von Linien, Rechtecken, Ovalen, Strings, Bildern und anderen Grafikprimitiven benötigen.

Unterstützt wird die Klasse `android.graphics.Canvas` dabei von der Klasse `android.graphics.Paint`, deren Objekte unsere Zeichenwerkzeuge repräsentieren und über die wir Farbe, Linienbreite und andere Parameter einstellen können.

### 9.1.4 Wie alles zusammenwirkt

Um einer Bildschirmseite eine View hinzuzufügen, in die Sie zeichnen können, müssen Sie zuerst Ihrem App-Projekt eine neue Quelldatei für Ihre abgeleitete View-Klasse hinzufügen. Danach bauen Sie die View in das Layout der Bildschirmseite ein und zum guten Schluss erweitern Sie die Definition der View-Klasse um Ihren Zeichencode.

Wenn Sie alles gleich parallel zum Lesen in Android Studio ausprobieren wollen, dann wäre es ein guter Zeitpunkt, um ein neues Projekt anzulegen. Verwenden Sie folgende Einstellungen:

**Tabelle 9.1** Parameter für das Projekt Grafik

Dialogfeld	Eingabe/Einstellung
Application name	Grafik
Domain name	standard.example.com
Target Android Device	Phone and Tablet, API 15
ADD AN ACTIVITY	Empty Activity

## Quelldatei anlegen

Am besten rufen Sie dazu in der Projektansicht unter dem *app/src/java*-Knoten das Kontextmenü im Paketknoten *com.example.standard.grafik* auf und wählen den Befehl **New/Java Class**. Geben Sie dann den Namen für die neu anzulegende Klasse ein, wie z. B. *ZeichnenView*. Nach dem Erzeugen wird der Code von Android Studio im Editor angezeigt. Wir müssen nun noch von der Basisklasse *android.widget.ImageView* ableiten, indem eine *extends*-Anweisung hinzugefügt und ein Default-Konstruktor bereitgestellt wird (Letzteres können Sie bequem mit **(Alt)+(Einf)** generieren lassen:

**Listing 9.1** Anlegen einer Quelldatei für eine View, in die man zeichnen kann

```
package com.example.standard.grafik;

import android.content.Context;
import android.widget.ImageView;

public class ZeichnenView extends ImageView {
    public ZeichnenView(Context context) {
        super(context);
    }
}
```

Grundsätzlich gibt es zwei Techniken, wie Sie Ihre Zeichenflächen-View in eine Bildschirmseite einbauen können:

- Sie bauen die View direkt in den XML-Code der Layoutdatei ein (im Ordner *app/res/layout* zu finden).
- Sie erzeugen die View programmatisch per Java-Code und bauen sie zur Laufzeit in das Layout ein.

### Variante 1: Eigene View-Klasse in XML-Layout verwenden

Im ersten Fall müssen Sie in die XML-Layoutdatei ein XML-Element einbauen, das die View repräsentiert. Da die View von einem selbst geschriebenen Typ ist, gibt es kein vordefiniertes XML-Element für diese View (und es taucht daher auch nicht in der **Design**-Ansicht in der Palette auf). Als Name des XML-Elements benutzen Sie daher den vollständigen Namen der View-Klasse (im Beispiel *com.example.standard.grafik.ZeichnenView*).

**Listing 9.2** Layoutdatei *activity\_main.xml* mit eigener View-Klasse

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.standard.grafik.MainActivity">

    <com.example.standard.grafik.ZeichnenView
        android:id="@+id/zeichnen">
```



```

    android:layout_width="match_parent"
    android:layout_height="250dp"
    android:background="#ffff00"
  />

```

```
</RelativeLayout>
```

Bei Ausführung der App wird für jedes in der Layoutdatei definierte XML-Element ein Objekt der zugehörigen Klasse erzeugt.

Bisher mussten wir uns um diesen Mechanismus nie weiter kümmern, weil er automatisch ablief und wir nur XML-Elemente zu vordefinierten Android-Klassen benutzten.

Wir können diesen Mechanismus auch für selbst geschriebene View-Klassen verwenden, müssen dann aber in der Klasse einen Konstruktor definieren, der als Parameter sowohl ein Context-Objekt als auch ein AttributeSet-Objekt (zur Übergabe der XML-Attribute) definiert:

```

package com.example.standard.grafik;

import android.content.Context;
import android.util.AttributeSet;
import android.widget.ImageView;

public class ZeichnenView extends ImageView {

    // wenn Klasse programmatisch in der Activity erzeugt wird
    public ZeichnenView(Context context) {
        super(context);
    }

    // wenn Klasse in XML-Layoutdatei verwendet wird
    public ZeichnenView(Context context, AttributeSet attrs) {
        super(context, attrs);
    }
}

```



### ACHTUNG

Wenn Sie keine Hintergrund-Bitmap in die View laden, sollten Sie im XML-Code eine definierte Breite und Höhe vorgeben (also nicht `wrap_content` als Wert verwenden).

Das via XML-Layout erzeugte Objekt können wir uns (wie wir es bisher in den vorangegangenen Kapiteln auch immer gemacht haben) in der `onCreate()`-Methode besorgen und für späteren Gebrauch in einer Variablen merken.

```

public class MainActivity extends AppCompatActivity {
    private ZeichnenView zview;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}

```

```

        zview = (ZeichnView) findViewById(R.id.zeichnen);
    }
}

```

## Variante 2: Eigene View-Klasse programmatisch erzeugen

Wenn man die View-Klasse zur Laufzeit erzeugen will, dann ist die `onCreate()`-Methode der Activity (`MainActivity`) der richtige Platz. Dort erzeugen Sie ein Objekt Ihrer View (in unserem Fall also `ZeichnView`), beschaffen sich einen Verweis auf die gewünschte Layout-View, wo sie eingebaut werden soll, und benutzen deren `addView()`-Methode, um Ihre View als letztes Element in die Layout-View einzufügen.

```

package com.example.standard.grafik;

import android.graphics.Color;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.RelativeLayout;

public class MainActivity extends AppCompatActivity {
    private ZeichnView zview;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        zview = new ZeichnView(this);
        RelativeLayout.LayoutParams params = new
            RelativeLayout.LayoutParams(
                RelativeLayout.LayoutParams.MATCH_PARENT, 250);
        zview.setLayoutParams(params);

        zview.setBackgroundColor(Color.YELLOW);

        RelativeLayout layout = (RelativeLayout) findViewById(R.id.layout1);
        layout.addView(zview);
    }
}

```



Die ID für die Layout-View müssen Sie eigenhändig in der XML-Layoutdatei vergeben, da Android Studio keine automatisch generiert, z. B.

```
<RelativeLayout xmlns:... android:id="@+id/layout1" > ...
```

Vergessen Sie übrigens nicht das `+`-Zeichen, da eine neue ID definiert wird (bei späterem Verweis bzw. Gebrauch der ID im XML-Code muss dann auf das `+` verzichtet werden).

## Zeichencode vorsehen

Um nun in die View zeichnen zu können, überschreiben Sie in Ihrer abgeleiteten View-Klasse die geerbte `onDraw()`-Methode. Sie können das Methodengerüst per Hand anlegen oder einfach in den Code der Klasse klicken, die Tastenkombination `Alt+Einf` drücken (oder

mit rechter Maustaste das Kontextmenü aufrufen und **Generate** auswählen) und dann den Eintrag **Override Methods** markieren. Suchen Sie dann die Methode `onDraw()` und klicken Sie auf **OK**.

**Listing 9.3** Überschreiben der `onDraw()`-Methode

```
package com.example.standard.grafik;

import android.content.Context;
import android.graphics.Canvas;
import android.util.AttributeSet;
import android.widget.ImageView;

public class ZeichnenView extends ImageView {
    public ZeichnenView(Context context) {
        super(context);
    }

    public ZeichnenView(Context context, AttributeSet attrs) {
        super(context, attrs);
    }

    @Override
    protected void onDraw(Canvas canvas) {
        super.onDraw(canvas);
    }
}
```

## ■ 9.2 Grafikprimitive zeichnen

Das eigentliche Zeichnen geschieht durch Aufruf der dafür vorgesehenen Canvas-Methoden.

Zuerst erzeugen Sie ein `Paint`-Objekt, über das Sie z.B. Farbe und gewünschte Linienbreite (für Linien und Umrisse) festlegen.

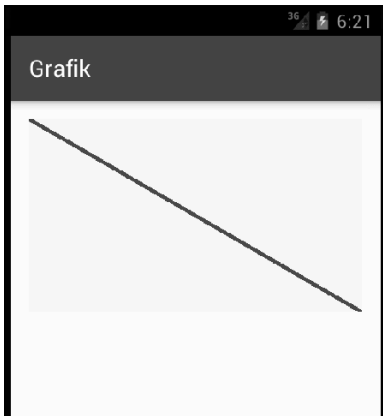
Dann wählen Sie abhängig von der Art des Grafikprimitivs (Linie, Rechteck, Kreis etc.), das Sie zeichnen möchten, die passende Canvas-Methode und schließlich übergeben Sie der Methode die Koordinaten und das `Paint`-Objekt, das die Methode zum Zeichnen des gewünschten Grafikprimitivs verwenden soll.

**Listing 9.4** Überschreiben der `onDraw()`-Methode

```
protected void onDraw(Canvas canvas) {
    super.onDraw(canvas);

    // 1. Paint-Objekt für Blauton und
    // Linienbreite von 5 Pixeln erzeugen
    Paint pinsel = new Paint();
    pinsel.setColor(Color.rgb(64, 64, 255));
    pinsel.setStrokeWidth(5);
}
```

```
// Diagonale durch Leinwand zeichnen
canvas.drawLine(0, 0, getWidth(), getHeight(), pinse1);
}
```

**Bild 9.1**

Bildschirmseite mit ImageView für eigene Grafikausgaben

**Tabelle 9.2** Auswahl einiger Methoden der Klasse Canvas

Methode	Beschreibung
<code>drawColor(int farbe)</code> <code>drawRGB(int r,int g,int b)</code>	Füllt die ganze Leinwand in der angegebenen Farbe (Color-Wert oder RGB-Anteile)
<code>drawArc(RectF r,</code> <code>float start,</code> <code>float winkel,</code> <code>bool keil,</code> <code>Paint pinse1)</code>	Zeichnet einen Bogen in das Rechteck <code>r</code> ein. Das Argument <code>start</code> gibt den Startwinkel an. Über das boolesche Argument können Sie festlegen, ob Sie nur einen Bogen ( <code>false</code> ) oder einen Keil ( <code>true</code> ) zeichnen möchten.
<code>drawBitmap(Bitmap b,</code> <code>float links,</code> <code>float oben,</code> <code>Paint pinse1)</code>	Kopiert ein Bild in das Canvas, sodass die linke obere Ecke des Bilds an der angegebenen Koordinate zu liegen kommt. Für das <code>Paint</code> -Objekt können Sie <code>null</code> übergeben oder ein Objekt, das eine Maske für die Verblendung mit dem Hintergrund definiert.
<code>drawBitmap(Bitmap b,</code> <code>RectF quelle,</code> <code>RectF ziel,</code> <code>Paint pinse1)</code>	Kopiert ein Bild oder einen Bildausschnitt (Argument <code>quelle</code> ungleich <code>null</code> ) in den angegebenen Zielbereich des Canvas. Das Bild wird notfalls skaliert, damit es in den Zielbereich passt. Für das <code>Paint</code> -Objekt können Sie <code>null</code> übergeben oder ein Objekt, das eine Maske für die Verblendung mit dem Hintergrund definiert.
<code>drawBitmap(Bitmap b, Matrix matrix,</code> <code>Paint pinse1)</code>	Kopiert das Bild an die durch eine Transformationsmatrix angegebene Stelle. <code>matrix</code> kann eine beliebige Kombination von Verschiebung/Rotation/Skalierung beinhalten.

(Fortsetzung nächste Seite)

**Tabelle 9.2** Auswahl einiger Methoden der Klasse `Canvas` (Fortsetzung)

Methodenname	Beschreibung
<code>drawCircle(float x, float y, float radius, Paint pinsel)</code>	Zeichnet einen Kreis mit Mittelpunkt <code>x,y</code> und dem angegebenen Radius.
<code>drawLine(float startX, float startY, float endX, float endY, Paint pinsel)</code>	Zeichnet eine Linie von <code>(startX,startY)</code> nach <code>(endX,endY)</code> .
<code>drawLines(float[] punkte, Paint pinsel)</code>	Zeichnet mehrere Linien. Für jede Linie müssen im Array <code>punkte</code> vier Werte angegeben werden.
<code>drawOval(RectF r, Paint pinsel)</code>	Zeichnet ein Oval in das vorgegebene Rechteck. Wenn Sie als erstes Argument ein Quadrat übergeben, erhalten Sie einen Kreis.
<code>drawPoints(float[] punkte, int offset, int anzahl, Paint pinsel)</code>	Zeichnet mehrere Punkte. Für jeden Punkt müssen im Array <code>punkte</code> zwei Werte ( <code>x,y</code> ) angegeben werden. Die ersten <code>offset</code> Punkte werden übersprungen und es werden maximal <code>anzahl</code> Punkte gezeichnet. Die Dicke der Punkte können Sie über den <code>Stroke</code> -Wert des Pinsels festlegen.
<code>drawRect(float links, float oben, float rechts, float unten, Paint pinsel)</code> <code>drawRect(RectF r, Paint pinsel)</code>	Zeichnet ein Rechteck.
<code>drawRoundRect(RectF r, float rx, float ry, Paint pinsel)</code>	Zeichnet ein Rechteck mit abgerundeten Ecken. Die Radien für die abgerundeten Ecken können Sie über die Argumente <code>rx</code> und <code>ry</code> vorgeben.
<code>drawText(String text, float x, float y, Paint pinsel)</code>	Zeichnet einen String an der spezifizierten Koordinate.
<code>fillArc()</code> , <code>fillOval()</code> , <code>fillPolygon()</code> ...	Zeichnet ausgefüllte Formen (vgl. <code>draw...()</code> ).
<code>scale(float fx, float fy)</code>	Skaliert das Canvas um den Faktor <code>fx</code> (für x-Achse) und <code>fy</code> (y-Achse).
<code>translate(float dx, float dy)</code>	Verschiebt den Ursprung des Canvas.

## Koordinaten

Koordinaten können Sie als `int`- oder `float`-Werte angeben.

Etliche Methoden arbeiten mit umschließenden Rechtecken (Bounding Box), in die das zu zeichnende Grafikprimitive eingepasst wird. Diese Rechtecke können Sie meist auch als `RectF`-Objekt angeben.

```
Paint pinsel = new Paint();
pinsel.setColor(Color.CYAN);
pinsel.setStrokeWidth(2);

RectF r = new RectF(10, 10, 50, 50);
canvas.drawRect(r, pinsel);
```



Der Ursprung des Canvas liegt immer in der linken oberen Ecke (Koordinaten = 0,0) und die x-Werte nehmen nach rechts hin zu, die y-Werte nach unten hin.

## Farben

Farben werden grundsätzlich durch `int`-Werte codiert. Einige Farben sind als statische Felder der Klasse `android.graphics.Color` vordefiniert: `Color.BLACK`, `Color.BLUE`, `Color.CYAN`, `Color.DKGRAY`, `Color.GRAY`, `Color.GREEN`, `Color.LTGRAY`, `Color.MAGENTA`, `Color.RED`, `Color.TRANSPARENT`, `Color.WHITE`, `Color.YELLOW`. Alle anderen Farben können Sie mithilfe der statischen Methode `rgb()` definieren:

```
Paint pinsel = new Paint();
pinsel.setColor(Color.rgb(64, 64, 255));
pinsel.setStrokeWidth(2);

RectF r = new RectF(10, 10, 50, 50);
canvas.drawRect(r, pinsel);
```

## Umrisse oder gefüllte Primitive

Mit den `draw`-Methoden können Sie sowohl Umrisslinien als auch ausgefüllte Grafikprimitive zeichnen. Standardmäßig werden ausgefüllte Grafikprimitive gezeichnet (`Style.FILL`). Um einen Umriss zu zeichnen, müssen Sie die `setStyle()`-Methode des verwendeten `Paint`-Objekts aufrufen und dieser die Konstante `Style.STROKE` übergeben.

```
RectF r = new RectF(10, 10, 50, 50);

Paint umrissPinsel = new Paint();
umrissPinsel.setColor(Color.YELLOW);
umrissPinsel.setStrokeWidth(2);
umrissPinsel.setStyle(Style.STROKE);

Paint fuellPinsel = new Paint();
fuellPinsel.setColor(Color.MAGENTA);
fuellPinsel.setStyle(Style.FILL);

canvas.drawRect(r, fuellPinsel);
canvas.drawRect(r, umrissPinsel);
```

## ■ 9.3 Bilder laden

Wenn Sie beispielsweise in einer View ein Hintergrundbild setzen wollen, dann reicht es, einen Aufruf der Art `setBackgroundResource(R.drawable.meinbild)` zu machen, und das war es schon. Android macht alle notwendigen Dinge.

Um eine Bildressource in einer App selbst zu verwenden, muss man leider etwas mehr Aufwand treiben, insbesondere, wenn man nicht weiß oder sicherstellen kann, dass das zu ladende Bild recht groß ist (einige Mbyte Platzbedarf). In solchen Fällen kommt es sehr schnell zu `OutOfMemory`-Fehlern, weil der verfügbare Speicher nicht ausreicht.

Das Laden von Bildern, die groß sind oder wo Sie es nicht wissen (können), sollten Sie daher immer in drei Schritten ausführen:

- Ermitteln Sie die Größe (Breite/Höhe) des zu ladenden Bildes. Dies erfordert ein provisorisches Laden des Bildes ohne die eigentlichen Bilddaten.
- Berechnen Sie einen geeigneten Skalierungsfaktor, damit es für die geplante Anzeige (eine View o. Ä.) passt.
- Laden Sie das Bild.

Codemäßig sehen die Schritte dann folgendermaßen aus. Zuerst lädt man das Bild mit der statischen Methode `BitmapFactory.decodeResource()` und übergibt dabei ein `BitmapFactory.Options`-Objekt, dessen Feld `inJustDecodeBounds` auf `true` gesetzt ist:

```
BitmapFactory.Options options = new BitmapFactory.Options();
options.inJustDecodeBounds = true;
BitmapFactory.decodeResource(getResources(), id, options);
```

Das `options`-Objekt enthält nun die Bildmaße in den Variablen `outWidth` und `outHeight`. Dann muss man den Anpassungsfaktor berechnen. Man startet mit 1 und, falls das Bild größer ist als der geplante Anzeigeort, wird er immer weiter verdoppelt, bis das Verhältnis Größe/Faktor klein genug geworden ist:

```
private int berechneLadefaktor(BitmapFactory.Options options,
                               int breiteAnzeige, int hoeheAnzeige){
    final int w = options.outWidth;
    final int h = options.outHeight;
    int faktor = 1;

    if (w > breiteAnzeige || h > hoeheAnzeige) {
        int w_halbe = w / 2;
        int h_halbe = h / 2;

        while((w_halbe / faktor) > breiteAnzeige
            && ((h_halbe / faktor) > hoeheAnzeige)) {
            faktor = faktor * 2;
        }
    }

    return faktor;
}
```

Nun kann man die Bilddaten laden. Wichtig ist dabei, den berechneten Faktor an die Variable `inSampleSize` zuzuweisen und das Flag `inJustDecodeBounds` wieder auf `false` zu stellen, z. B.

```
options.inSampleSize = berechneLadefaktor(options,
                                         viewBreite, viewHoehe);
options.inJustDecodeBounds = false;
Bitmap bild = BitmapFactory.decodeResource(getResources(),
                                     R.drawable.meinbild, options);
```

## ■ 9.4 In Bilder hineinzeichnen

Bilder, die man als `Bitmap`-Objekte geladen hat, kann man nicht einfach nur anzeigen. Man kann sie auch mit eigenen Zeichenoperationen verändern. Das ist oft einfacher und vom Ergebnis her viel ansprechender, als alles komplett mithilfe der Zeichenmethoden der Klasse `Canvas` selbst zu erstellen. Hierzu muss man einfach ein eigenes `Canvas`-Objekt erzeugen, wobei man dem Konstruktor das `Bitmap` mitgibt. Allerdings lauert hier eine kleine Falle: Wenn das `Bitmap`-Objekt durch das Laden einer Bildressource erzeugt worden ist, dann es ist nicht veränderbar (*immutable*). Man muss daher zuvor eine Kopie anlegen und kann dann darauf ein `Canvas`-Objekt erstellen:

```
BitmapFactory.Options options = new BitmapFactory.Options();
Bitmap origBild = BitmapFactory.decodeResource(getResources(),
                                           R.drawable.meinbild, options);

// Kopie erzeugen
Bitmap kopie = origBild.copy(origBild.getConfig(), true);

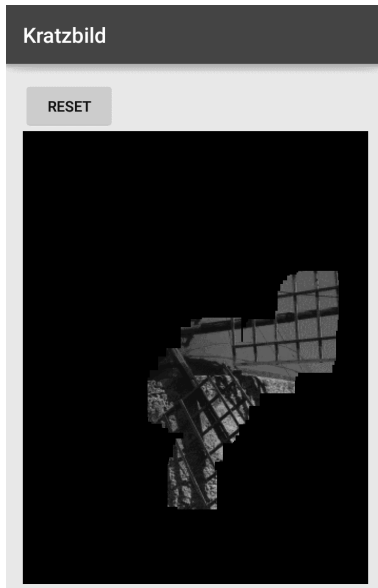
// Canvas zum Zeichnen
Canvas canvasKopie = new Canvas(kopie);
canvasKopie.drawRect(...) // usw.
```

Nun hat man ein `Canvas`-Objekt und kann auf der Bildkopie zeichnen. Alle Veränderungen werden sichtbar, wenn das `Bitmap`-Objekt (hier `kopie`) via `drawImage()` in einer `View.onDraw()`-Methode gerendert wird, z. B.

```
@Override
protected void onDraw(Canvas canvas) {
    super.onDraw(canvas);
    canvas.drawBitmap(kopie, 0, 0, null);
}
```

Als kleine Veranschaulichung finden Sie unter <http://files.hanser.de/fachbuch/Beispiele.zip> zu diesem Kapitel das Beispielprojekt *Kratzbild*. Hierbei wird mit zwei `Bitmaps` gearbeitet: Eins dient zur Darstellung eines Fotos, das andere ist zu Beginn einfach schwarz und verdeckt das Foto. Der Benutzer kann es durch Fingerbewegung freikratzen und das Foto kommt nach und nach zum Vorschein.





**Bild 9.2**  
Beispiel-App Kratzbild

Wir betrachten nur die zwei bzgl. des Zeichnens interessanten Codeteile.

Zum einen wird eine neue Bitmap mithilfe der statischen Methode `Bitmap.createBitmap()` in der Größe der Anzeigefläche erstellt. Auf dieser Bitmap erzeugen wir nun ein `Canvas`-Objekt und zeichnen darin mit einem `drawRect()`-Befehl ein alles bedeckendes schwarzes Rechteck:

```
Bitmap kratzSchicht = Bitmap.createBitmap(anzeigeBreite,
                                          anzeigeHoehe, Bitmap.Config.ARGB_8888);
Canvas kratzSchichtCanvas = new Canvas(kratzSchicht);
Rect r = new Rect(0,0, kratzSchicht.getWidth(),
                 kratzSchicht.getHeight());
Paint paint = new Paint();
paint.setColor(Color.BLACK);
kratzSchichtCanvas.drawRect(r, paint);
```

In der `onDraw()`-Methode der View wird immer zuerst das anzuzeigende Bild (Bitmap `kratzbild`) gezeichnet und dann darüber die schwarze Kratzschicht. Zu Beginn sieht der Benutzer also nur schwarz:

```
@Override
protected void onDraw(Canvas canvas) {
    super.onDraw(canvas);

    // ...
    canvas.drawBitmap(kratzbild, offsetX, offsetY, null);

    canvas.drawBitmap(kratzSchicht, 0, 0, null);
}
```

Wenn der Benutzer nun mit dem Finger herumkratzt, fangen wir die Mausbewegungen ab und machen an dieser Stelle die Kratzschicht durchsichtig, d. h., wir müssen dem Canvas kratzschichtCanvas den Befehl erteilen, an einer bestimmten Stelle schwarz durch transparent zu ersetzen. Hierzu brauchen wir ein Paint-Objekt mit Transfer-Modus<sup>1</sup> CLEAR:

```
Paint transparentPaint = new Paint();
transparentPaint.setXfermode(new
    PorterDuffXfermode(PorterDuff.Mode.CLEAR));
```

Bei jedem Mausevent müssen wir dann nur noch auf der kratzschichtCanvas an der aktuellen Stelle ein „Loch brennen“ und per invalidate() das Neuzeichnen veranlassen.

```
public void updateMouse(int x, int y) {
    kratzschichtCanvas.drawRect(x, y, x + 75, y + 75,
        transparentPaint);
    invalidate();
}
```

## ■ 9.5 Bilder bewegen

In diesem Abschnitt möchten wir Ihnen zeigen, wie man es einrichten kann, dass der Anwender eine Figur (ein UFO) mithilfe der Finger über den App-Hintergrund bewegt.



**Bild 9.3**  
Eine fliegende Untertasse

<sup>1</sup> Leider sprengt die Komplexität von Computergrafik den Rahmen dieses Buches, um dies und vieles anderes weiter zu erläutern.

Dazu benötigen wir:

- eine App mit einem `FrameLayout` und einer selbst definierten View
- eine Bitmap für das UFO vor transparentem Hintergrund
- Code zum Zeichnen des UFO
- Code zur Behandlung der Richtung

Falls Sie die Schritte parallel zum Lesen in Android Studio nachverfolgen wollen, dann sollten Sie ein neues Projekt mit den üblichen Einstellungen anlegen:

**Tabelle 9.3** Parameter des Projekts UFO

Dialogfeld	Eingabe/Einstellung
Application name	UFO
Domain name	standard.example.com
Target Android Device	Phone and Tablet, API 15
ADD AN ACTIVITY	Empty Activity

### View-Klasse, Activity und Layoutdatei

In Android Studio neu angelegte App-Projekte verfügen (bei Erstellung einer Empty Activity) immer über eine Start-Activity mit einem `RelativeLayout`. Für Apps, die wie unsere UFO-App nur aus einer einzigen Zeichenfläche bestehen (also einem einzigen anzuzeigenden View-Element), ist das `FrameLayout` besser geeignet. Der erste Schritt besteht also darin, das `RelativeLayout` durch `FrameLayout` zu ersetzen.

**Listing 9.5** Die Layoutdatei `activity_main.xml` (aus dem Projekt UFO)

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:id="@+id/frameLayout0"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
</FrameLayout>
```



Um die oberste Layout-View auszutauschen, öffnen Sie die Layoutdatei `activity_main.xml` am besten in der XML-Ansicht.

Als Zeichenfläche verwenden wir eine selbst geschriebene View-Klasse, die wir direkt von der Basisklasse `View` ableiten. Im Konstruktor dieser Klasse laden wir die Bildressource für den Hintergrund. Hierzu haben wir ein geeignetes Foto als Datei `hintergrund.png` im Projektordner `app\src\main\res\drawable` abgelegt.

**Listing 9.6** Definition der Zeichenflächen-View (aus UFOView.java)

```
package com.standard.example.ufo;

import android.content.Context;
import android.content.res.Resources;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Canvas;
import android.graphics.Paint;
import android.view.MotionEvent;
import android.view.View;

public class UFOView extends View {
    public UFOView(Context c) {
        super(c);

        // lade Hintergrund
        this.setBackgroundResource(R.drawable.hintergrund);
    }
}
```

In der `onCreate()`-Methode der Activity-Klasse erzeugen wir nun ein Objekt unserer View-Klasse und fügen dieses programmatisch durch Java-Code zur Laufzeit in das `FrameLayout` ein.

**Listing 9.7** Einbetten der Zeichenflächen-View (aus MainActivity.java)

```
package com.standard.example.ufo;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.FrameLayout;

public class MainActivity extends AppCompatActivity {
    private UFOView ufoView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // UI vervollständigen
        ufoView = new UFOView(this);

        FrameLayout frameLayout =
            (FrameLayout) findViewById(R.id.frameLayout1);
        frameLayout.addView(ufoView);
    }
}
```

## Die UFO-Bitmap

Bitmaps haben immer rechteckige Abmaße. Wenn Sie daher Bitmaps für Figuren erzeugen, die Sie über einen Hintergrund oder zwischen anderen Figuren bewegen möchten, müssen

Sie darauf achten, dass Sie die Figur vor einem einfarbigen Hintergrund zeichnen und diesen dann als transparent deklarieren. Mit etwas besser ausgestatteten Grafikprogrammen sollte dies kein Problem sein.



### SPRITES

Figuren, die bewegt werden, bezeichnet man auch als *Sprites*. Speichern Sie Ihre Sprites im PNG-Format. (Das JPG-Format unterstützt keine Transparenz.)

Speichern Sie die Bitmap der Figur als Ressource *ufo.png* (Ordner *app/src/main/res/drawable*). Laden Sie die Bitmap-Ressource in der *onCreate()*-Methode der View-Klasse in ein Bitmap-Objekt. Verwenden Sie dazu die *BitmapFactory*-Methode *decodeResource()*.

```
public class UFOView extends View {
    private float xpos = -1;
    private float ypos = -1;
    private Bitmap ufoBitmap;

    public UFOView(Context c) {
        super(c);

        // lade Hintergrund
        this.setBackgroundResource(R.drawable.hintergrund);

        // lade Bitmap für UFO
        Resources resources = getResources();
        ufoBitmap = BitmapFactory.decodeResource(resources,
                                                R.drawable.ufo);
    }
}
```

Haben Sie die Felder *xpos* und *ypos* bemerkt? Sie sollen die Position (Mittelpunkt) angeben, an der die UFO-Bitmap in das Canvas gezeichnet werden soll.

Anfangs haben diese Koordinaten den Wert *-1*, obwohl wir das UFO eigentlich in der Mitte der View anzeigen möchten. Die Methoden, die uns Breite und Höhe der View liefern (*getWidth()* und *getHeight()*), geben allerdings in der *onCreate()*-Methode noch keine vernünftigen Werte zurück (auch nicht in *onStart()*). Wir starten daher mit den Anfangswerten *-1* und korrigieren diese beim ersten Aufruf von *onDraw()*. Später werden die Koordinaten dann nur noch beim Tippen durch den Anwender verändert.

### onDraw()

In der *onDraw()*-Methode kopieren wir die Bitmap in das Canvas. Als Zielposition wählen wir die Koordinate (*xpos*, *ypos*), wobei wir von dem *x*-Wert die halbe Bitmap-Breite und von dem *y*-Wert die halbe Bitmap-Höhe abziehen, damit die Bitmap ungefähr mittig über dem Punkt (*xpos*, *ypos*) zu liegen kommt.

```
protected void onDraw(Canvas canvas) {
    super.onDraw(canvas);

    // Startposition beim ersten Aufruf auf View-Mitte setzen
```

```

if (xpos == -1 && ypos == -1) {
    xpos = getWidth()/2;
    ypos = getHeight()/2;
}

// UFO-Bitmap einzeichnen
Paint iconPaint = new Paint(Paint.ANTI_ALIAS_FLAG |
                             Paint.DITHER_FLAG |
                             Paint.FILTER_BITMAP_FLAG
);

if(ufoBitmap != null) {
    canvas.drawBitmap(ufoBitmap,
                      xpos - ufoBitmap.getWidth()/2,
                      ypos - ufoBitmap.getHeight()/2,
                      iconPaint);
}
}

```

Achten Sie insbesondere darauf, dass Sie für das Zeichnen eines Bitmaps ein Paint-Objekt benutzen sollten, das mit verschiedenen Schaltern (*Flags*) zum optimierten Zeichnen von Bitmaps initialisiert worden ist (`Paint.ANTI_ALIAS_FLAG | Paint.DITHER_FLAG | Paint.FILTER_BITMAP_FLAG`).

Beim ersten Aufruf von `onDraw()` korrigieren wir die Werte von `xpos` und `ypos` so, dass das UFO in der Mitte der View angezeigt wird.

### **onTouchEvent()**

Um sicherzustellen, dass unsere Zeichenflächen-View überhaupt Ereignisse empfängt, rufen wir in ihrer `onCreate()`-Methode die Methode `setFocusable()` auf:

```

public UFOView(Context c) {
    super(c);

    // ...

    setFocusable(true);
}

```

Der Anwender soll das UFO durch Tippen mit dem Finger steuern. Wir brauchen also eine Behandlung von Touch-Ereignissen. Hierbei haben wir zwei Möglichkeiten:

- Wir registrieren bei der `UFOView` einen eigenen `OnTouchListener` oder
- wir überschreiben die Methode `onTouchEvent()` in der `UFOView`-Klasse

Beide Ansätze sind absolut gleichwertig und wir nehmen mal die zweite Variante und überschreiben die `onTouchEvent()`-Methode:

```

@Override
public boolean onTouchEvent(MotionEvent event) {
    int action = event.getAction();

    float dx = event.getX() - xpos;
    float dy = event.getY() - ypos;
}

```

```

    if(action == MotionEvent.ACTION_MOVE) {
        xpos += dx;
        ypos += dy;
    }
    else if(action == MotionEvent.ACTION_DOWN ) {
        xpos += Math.signum(dx) * 25;
        ypos += Math.signum(dy) * 25;
    }

    if(xpos < 0) {
        xpos = 0;
    }
    if(xpos > getWidth()) {
        xpos = getWidth();
    }
    if(ypos < 0) {
        ypos = 0;
    }
    if(ypos > getHeight()) {
        ypos = getHeight();
    }

    invalidate();
    return true;
}

```

Wurde eine Berührung (ACTION\_DOWN) oder ein Wischen des Fingers (ACTION\_MOVE) ausgeführt, dann ermitteln wir die Distanz zwischen der Stelle, wo der Finger berührt, und der aktuellen Mitte des UFO. Das ergibt die Information, in welche Richtung sich das UFO bewegen soll. Im Falle eines einzelnen Tippens verschieben wir das UFO pauschal um 25 Pixel, bei der Dauerberührung um die aktuelle Pixeldifferenz.

Zum Schluss erfolgt noch eine Prüfung, ob die UFO-Position in Gefahr gerät, den Bildschirm zu verlassen. In solchen Fällen stoppen wir das UFO an der jeweiligen Grenze.

Am Ende der Methode rufen wir `invalidate()` auf. Der `invalidate()`-Aufruf bewirkt, dass die `UFOView` ein Signal erhält, dass sie sich selbst neu zeichnen soll, d. h., es wird im Endeffekt `onDraw()` ausgeführt und dabei werden die neuen Werte von `xpos` und `ypos` verwendet.

## ■ 9.6 Verbesserungen

Die hier vorgestellten Techniken eignen sich allerdings nur für einfache Grafikanwendungen. Wenn Sie für die Aktualisierung der gezeichneten Oberfläche länger andauernde Berechnungen durchführen müssen oder mehrere Objekte dynamisch und schnell verschieben möchten, lagern Sie den Code zum Zeichnen in einen Thread aus, verwenden Sie `SurfaceView` und `SurfaceHolder.Callback` und führen Sie alle Zeichenoperationen im Hintergrund auf einem eigenen Canvas aus, den Sie dann erst nach Abschluss der Zeichenoperationen in einem Rutsch in den Canvas der View kopieren.

## ■ 9.7 Fragen und Antworten

1. *Irgendwo habe ich einmal gewundenen Text gesehen. Wie haben die das gemacht?*

Vermutlich mit der Canvas-Methode `drawTextOnPath()`. Hier wird ein String entlang eines vorgegebenen Pfads (Klasse `android.graphics.Path`) gezeichnet.

2. *Wie kann man halb durchsichtig (transparent) zeichnen?*

Transparenz wird im RGB-Farbmodell durch den sogenannten Alpha-Kanal festgelegt. Die `Paint`-Klasse bietet daher die Methode `setAlpha()`, die einen Wert zwischen 0 (völlig transparent) und 255 (undurchsichtig) erwartet.

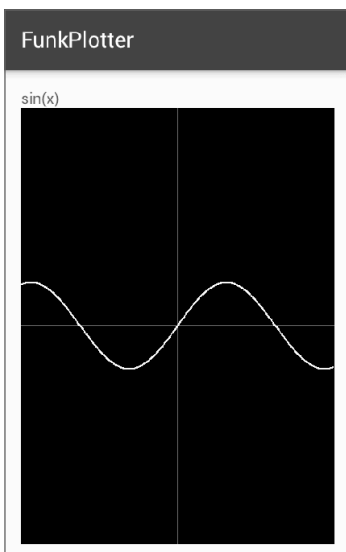
3. *Moderne Programmiersprachen bieten auch Gradientenfüllung. Was ist mit Android?*

Gibt's natürlich auch. Man kann z. B. die Klasse `LinearGradient` verwenden, um einen linearen Farbverlauf von einem Startpunkt zu einem Endpunkt und von Gelb zu Blau zu erreichen:

```
LinearGradient lg = new LinearGradient(startx, starty,  
                                     endx, endy,  
                                     Color.YELLOW, Color.BLUE, TileMode.CLAMP);  
  
Paint pinsel = new Paint();  
pinsel.setShader(lg);
```

## ■ 9.8 Übung

Schreiben Sie einen Funktionsplotter, der für die Funktion  $f(x) = \sin(x)$  den Verlauf im Wertebereich  $-5$  bis  $+5$  zeichnet.



**Bild 9.4**  
Ausgabe von  $\sin(x)$



# Index

## Symbole

@Override 30

## A

ActionBar 212, 215

Action-Item 215

Action-Menü 212

Activities 21, 23, 29, 50

- beenden 256

- Ergebnisse zurücksenden 256, 268

- Manifestdatei 255

- Start-Activity 58

- starten 250

Activity

- fileList() 263

- findViewById() 130

- finish() 184, 256

- getFilesDir() 263

- getIntent() 253

- getResources() 139

- getSystemService() 301

- onContextItemSelected() 221

- onCreate() 30

- onCreateContextMenu() 218

- onCreateDialog() 234

- onCreateOptionsMenu() 215 f.

- onOptionsItemSelected() 221

- openFileInput() 261

- openFileOutput() 261

- registerForContextMenu() 218

- setContentView() 30, 32, 128

- startActivity() 252

- startActivityForResult() 268

Activity-Menü 211

Adapter 107

- ArrayAdapter 365

- BaseAdapter 358

- Bilddaten 230

- SimpleCursorAdapter 325

AdapterContextMenuInfo 222

adb 396

addView() (ViewGroup) 195

ADM 186, 396

- Devices-Fenster 397

- Emulator Control-Fenster 400

- File Explorer-Fenster 399

- LogCat-Fenster 398

- LogCat-Filter anlegen 398

- starten 397

AlertDialog 235

android

- alpha 90

- background 90, 109

- checkedButton (RadioGroup) 117

- checked (CheckBox) 115

- checked (RadioButton) 117

- checked (Switch) 117

- checked (ToggleButton) 118

- columnCount (GridLayout) 105

- contentDescription 121

- contentDescription (ImageButton) 116

- contentDescription (ImageView) 116

- focusable 121

- gravity (LinearLayout) 100

- id 90

- inputType (EditText) 116

- layout\_above (RelativeLayout) 103

- layout\_align... (RelativeLayout) 103

- layout\_below (RelativeLayout) 103

- layout\_center... (RelativeLayout) 103

- layout\_columnWidth (GridView) 108

- layout\_gravity (GridView) 108

- layout\_gravity (LinearLayout) 101

- layout\_height 97

- layout\_horizontalSpacing (GridView) 108

- layout\_marginBottom 99

- layout\_marginLeft 99

- layout\_marginRight 99

- layout\_marginTop 99

- layout\_numColumns (GridView) 108

- layout\_stretchMode (GridView) 108

- layout\_toLeftOf (RelativeLayout) 103

- layout\_toRightOf (RelativeLayout) 103

- layout\_verticalSpacing (GridView) 108

- layout\_weight (LinearLayout) 101

- layout\_width 97

- max (ProgressBar) 116

- minLines (EditText) 115

- onItemSelected (Spinner) 117

- orientation (LinearLayout) 100

- orientation (RadioGroup) 117

- padding 90

- password (EditText) 116

- progress (ProgressBar) 116

- prompt (Spinner) 117

- rotationX 90

- rowCount (GridLayout) 105

- scaleType (ImageView) 116

- src (ImageButton) 116

- src (ImageView) 116

- style (ProgressBar) 116

- text (Button) 115

- text (CheckBox) 115

- text (EditText) 115

- textOff (Switch) 117

- textOff (ToggleButton) 118

- textOn (Switch) 117

- textOn (ToggleButton) 118

- text (RadioButton) 117

- textSize (TextView) 118

- textStyle (TextView) 118
  - text (Switch) 117
  - text (TextView) 118
  - typeface (TextView) 118
  - visibility 90
  - Android
    - Google Play 373, 397
    - Hilfsmittel 3
    - Market Place 373
    - Plattformen 10
    - Referenz der API 12
    - SDK 3
    - SDK-Manager 9
    - Versionsnummern 15
  - Android-Architektur 177
  - Android-Bibliothek 54
  - Android Device Monitor 186, 396
  - Android Monitor 42
  - android.permission.ACCESS\_COARSE\_LOCATION 333
  - android.permission.ACCESS\_FINE\_LOCATION 333
  - android.permission.VIBRATE 353
  - Android SDK
    - Dokumentation 12
    - Unterverzeichnisse 12
  - Android Studio 3, 383
    - API-Dokumentation anzeigen 71
    - Apps bauen 37
    - Apps erstellen 37
    - Apps exportieren 386
    - Code Folding 65
    - Code-Generierung 76
    - Dateien suchen 71
    - Dateiverlauf 81
    - Dialogfeld New Project 18
    - Eclipse-Projekte importieren 385
    - geerbte Methoden überschreiben 77
    - Hierarchie-Ansicht 94
    - Installation 5
    - Klammerpaare identifizieren 74
    - Klasse anlegen 81
    - Klassendefinition suchen 71
    - Klassen-Informationen 71
    - Live Templates 75
    - Local History 81
    - Methoden-Informationen 71
    - Probleme bei der Ausführung 385
    - Projektansicht 25
    - Projektansicht aktualisieren 135
    - Projekte anlegen 18, 383
    - Projekte ausführen 384
    - Projekte löschen 385
    - Projekt in Explorer 26
    - Properties-Fenster 386
    - Quelldateien laden 56
    - Refactoring 78
  - Run-Konfigurationen 385
  - Surround 75
  - Syntaxfehler beseitigen 69
  - Syntaxhervorhebung 64
  - Verwendungen (Usages) suchen 73
  - Wörterbuch 386
  - Zeichenkette global suchen 73
  - Zeichenkette suchen 73
  - Zeilennummern 74
  - API 9
    - Bezug zu Android-Version 15
    - Referenz 12
  - APK-Datei 59, 179
  - App-Bar 212
  - AppBar 215
  - AppCompatActivity 234
  - Apps
    - Activities 21, 23, 29, 50
    - Android-Bibliothek 28
    - an Gerätekonfiguration anpassen 152
    - Anwendungsname 20
    - APK-Datei 59
    - Application Not Responding-Meldung 54
    - bauen (Build) 37
    - beenden (finish()) 184
    - Benutzeroberfläche 85, 118
    - Bildschirmseiten 50
    - Compile-SDK 21
    - deinstallieren 396
    - Ereignisse 157
    - erstellen (Build) 37
    - exportieren 376, 386
    - Galerien 107
    - Grundgerüst 26
    - Hoch- und Querformat 126
    - Intents 51
    - Komponenten 54
    - Launcher Icon 127
    - Launcher-Icon ändern 48
    - Layout 31
    - Layoutdatei 32
    - Manifestdatei 58
    - mehrsprachige 367
    - Minimum SDK-Version 21
    - Paket 20, 28, 56
    - Portrait/Landscape 126
    - Präferenzen 259
    - Projekt anlegen 18
    - Projektname 20
    - Ressourcen 34, 133
    - Ressourcendateien 57, 135
    - R.java 32, 56, 136, 138
    - R-Klasse 32
    - Screenshots für die Veröffentlichung 397
    - signieren 376
    - Strings 34
    - strings.xml 34
    - Target-SDK 21
    - testen, auf Smartphone 45
    - testen, im Emulator 38
    - veröffentlichten 373
    - Views 52
    - weitergeben 373
    - zeitraubende Operationen 54
    - Zugriff auf Dateisystem 260
    - Zugriff auf SD-Karte 264
  - App-Symbol 127
  - Arbeitsthread 241
  - ArrayAdapter 365
  - ART 4
  - ART Virtual Machine 178
  - assets 155
  - AsyncTask 348
    - doInBackground() 349
    - execute() 348
  - Attribute, *siehe auch* android
    - 89
    - allgemeine 90
    - Layoutparameter 96
    - Namespace 90
    - style 148
  - Audio
    - MediaPlayer 284
    - Ressourcen 281
    - SoundPool 282
    - Töne abspielen 290
  - AudioTrack 291
  - Außenabstand (Margin) 99
  - AVD 390
    - einrichten 38
- ## B
- Back-Stack 180
    - Fragments 226
  - BaseAdapter 358
    - getCount() 359
    - getItem() 359
    - getView() 358
  - Bauen 37
  - Beispiele
    - Bildergalerie 357
    - Funktionsplotter 209
    - Geolokation 331
    - Kratzbild 201
    - Quiz-App 273
    - Reaktions-App 267
    - Sensoren 299
    - TicTacToe-App 341
    - UFO-App 203
    - von der Website 410
  - Benutzeroberfläche 85

- Benutzeroberflächen
    - Design 118
    - erleichterte Bedienbarkeit 121
    - Hoch- und Querformat 126
    - Layout-Views 95
    - Widgets (Steuerelemente) 114
  - Berechtigungen
    - android.permission.ACCESS\_COARSE\_LOCATION 333
    - android.permission.ACCESS\_FINE\_LOCATION 333
    - android.permission.VIBRATE 353
  - Beschleunigungssensor 305
  - Bilder 145, 292
    - Bildergalerien 357
    - Formate 146
    - Größe 146
    - hineinzeichnen 201
    - Hintergrundbilder 111
    - laden 200
    - per Code laden 292
    - zeichnen 203
  - Bildergalerien 357
  - Bildschirmdichte 128
  - Bildschirmseiten 50
    - Design 85, 118
    - Hierarchie 94
    - Hoch- und Querformat 126
    - Layout-Views 95
    - Portrait/Landscape 126
    - View-Elemente 52
    - Widgets (Steuerelemente) 114
    - Wurzelement 88
    - XML-Code 88
  - Bitmap 292
  - BitmapFactory 292
    - decodeResource() 292
  - Breakpoint 402
  - Broadcast 257
  - Broadcast Intents 52, 257
  - Broadcast Receiver 53
  - Buch-DVD 4
    - Beispiele 409
  - Buch-Material 409
  - Buch-Website 14
  - Build (Bauen) 37
  - Bundle 252f.
  - Button 115
    - onClick 115
    - text 115
- C**
- Calendar 238
  - Callback 237
  - Canvas 191
    - drawBitmap() 197, 207
    - drawCircle() 198
    - drawColor() 197
    - drawLine() 198
    - drawLines() 198
    - drawOval() 198
    - drawPoints() 198
    - drawRect() 198
    - drawRoundRect() 197
    - drawText() 198
    - fill...() 198
    - scale() 198
    - translate() 198
  - CheckBox 115
    - checked 115
    - isChecked() 115
    - text 115
  - Class-Literal 252
  - close() (SQLiteDatabase) 319
  - Color 199
  - Compile-SDK
    - eines Projekts 21
    - nachträglich ändern 47
  - Content Provider 53, 329
  - ContentValues 321
  - convert() (Location) 335
  - create() (MediaPlayer) 284
  - Cursor 322
    - getCount() 322
    - getInt() 323
    - getString() 323
    - moveToFirst() 323
- D**
- Dalvik 4
  - Dalvik Virtual Machine 178
  - Dateien 260
    - assets 155
    - auf SD-Karte 264
    - lesen 261
    - Ressourcen 264
    - schreiben 261
    - Textdateien 262
  - Daten 259
    - als Preferences speichern 259
    - Persistenz 259
  - Datenbanken
    - als Ressourcen 319
    - anlegen 316
    - Datensatz 315
    - Datensätze aktualisieren 324
    - Datensätze einfügen 321
    - Datensätze lesen 322
    - Datensätze löschen 324
    - Fremdschlüssel 316
    - Groß- und Kleinschreibung 319
    - öffnen 316
    - Primärschlüssel 316, 318
    - relationale 315
    - schließen 319
    - SQL 316
    - SQLite 315
    - Treiber 316
  - DatePickerDialog 237
  - Datum, Auswahl über Dialog 237
  - DDMS 396
  - Debugging
    - ADM 396
    - Debugger 401
    - Haltepunkte 403
    - Logausgabe 184
    - starten 401
    - Variablen inspizieren 404
  - decodeResource() (BitmapFactory) 292
  - delete() (SQLiteDatabase) 324
  - Designer
    - Endgeräte simulieren 95
    - UI-Elemente ausrichten 92
    - UI-Elemente konfigurieren 94
    - UI-Hierarchie 94
  - Dialog 233
  - Dialoge 233
    - AlertDialog 235
    - anzeigen 235
    - eigene 242
    - erzeugen 234
  - DialogFragment 224, 234
    - getActivity() 235
    - onCreateDialog() 234
    - onCreateView() 234
  - distanceBetween() (Location) 335
  - distanceTo() (Location) 335
  - doinBackground() (AsyncTask) 349
  - DPAD 121, 392
  - Drawable 192
  - drawBitmap() (Canvas) 197
  - drawCircle() (Canvas) 198
  - drawColor() (Canvas) 197, 207
  - drawLine() (Canvas) 198
  - drawLines() (Canvas) 198
  - drawOval() (Canvas) 198
  - drawPoints() (Canvas) 198
  - drawRect() (Canvas) 198
  - drawRGB() (Canvas) 197
  - drawRoundRect() (Canvas) 198
  - drawText() (Canvas) 198
  - drawTextOnPath () (Canvas) 209
  - DVD, zum Buch 4, 409
- E**
- EditText 115
    - getText() 116
    - inputType 116
    - minLines 115

- password 116
- text 115
- Emulator 38, 389
- AVD einrichten 38
- AVD-Gerät 390
- Hoch- und Querformat 127
- SD-Karte 390
- Startoptionen 393
- zurücksetzen 394
- encode() (Uri) 286
- Environment 266
- Erdanziehung 306
- Ereignisse 157
  - Activity-Klasse 166
  - anonyme Listener-Klassen 165
  - anonyme Listener-Objekte 166
  - Behandlungscode einrichten 158
  - Gesten 174
  - Klickereignisse 158
  - Listener-Interfaces 158, 161 f.
  - Listener-Methoden implementieren 159
  - Listener-Objekt registrieren 159
  - Menüs 221
  - Multi-Touch 172
  - OnClickListener 158, 161
  - OnDragListener 161
  - onFocusChangeListener 162
  - OnKeyListener 162
  - OnLongClickListener 162
  - onTouchListener 162, 168
  - Sender ermitteln 167
  - Spinner 366
  - Tastaturereignisse 207
  - Tippereignisse 168
  - View-Parameter 167
  - Wischereignisse 170
- Erstellen 37
- execSql() (SQLiteDatabase) 318
- execute() (AsyncTask) 348
- Exportieren
  - Apps 386
- externer Speicher 260

## F

- Farben 109, 141, 199
- Fehlermeldungen
  - beheben 38
- FileInputStream 261
- fileList() (Activity) 263
- FileOutputStream 261
- fill...() (Canvas) 198
- fill\_parent 97
- Filter 306
  - Hochpass 308
  - Tiefpass 307
- findViewById() (Activity) 130

- finish() (Activity) 184, 256
- Fokus
  - Views 121
- Folding 65
- Fotos 295
- Fragment 224
- FragmentManager 225
- Fragments 53, 223
  - Back-Stack 226
- FragmentTransaction 226
- FrameLayout 108

## G

- Geokoordinaten
  - dezimal 334
  - sexagesimal 334
- Geolokation 331
  - Daten empfangen 332
  - Empfänger abmelden 333
  - GPS 331
  - Netzwerk 331
  - Provider 331
  - Verfügbarkeit 331
- Gesten 174
- getAccuracy() (Location) 340
- getAction() (MotionEvent) 169
- getActivity() (DialogFragment) 235
- getAltitude() (Location) 335
- getBearing() (Location) 335
- getCount() (BaseAdapter) 359
- getCount() (Cursor) 322
- getExternalStorageDirectory() (Environment) 266
- getFilesDir() (Activity) 263
- getInt() (Cursor) 323
- getIntent() (Activity) 253
- getItem() (BaseAdapter) 359
- getItemId() (MenuItem) 221
- getLatitude() (Location) 334
- getLongitude() (Location) 334
- getMenuInfo() (MenuItem) 222
- getReadableDatabase() (SQLiteOpenHelper) 317
- getResources() (Activity) 139
- getSensorList() (SensorManager) 301
- getSpeed() (Location) 335
- getString() (Cursor) 323
- getSystemService() (Activity) 301
- getText() (EditText) 116
- getTime() (Location) 334
- getView() (BaseAdapter) 358
- getWritableDatabase() (SQLiteOpenHelper) 317
- getX() (MotionEvent) 171
- getY() (MotionEvent) 171
- Gliederung 65

- Google Play 373
- GPS 331
- GPX 338
- Gradientenfüllung 209
- Grafik 191
  - Bilder zeichnen 203
  - Canvas 191
  - Farben 199
  - Füllung 199
  - Koordinaten 199
  - onDraw() 191, 195
  - Sprites 203
  - Umrisse 199
  - Zeichenwerkzeuge 192
  - zeichnen 196
- Gravitation
  - Somigliana 306
  - Vektor ermitteln 309
- Gravity (LinearLayout) 100
- GridLayout 105
  - columnCount 105
  - rowCount 105
- GridView 107, 357
  - layout\_columnWidth 108
  - layout\_gravity 108
  - layout\_horizontalSpacing 108
  - layout\_numColumns 108
  - layout\_stretchMode 108
  - layout\_verticalSpacing 108
- Größenangaben 98, 140
- Groß- und Kleinschreibung
  - Datenbanken 319
  - Klassennamen 29
- GUI
  - *siehe* Benutzeroberflächen

## H

- Haltepunkte 402
- Handler 244, 354
  - handleMessage() 356
  - sendMessage() 355
  - sendMessageDelayed() 356
- hasAccuracy() (Location) 340
- HAXM 42
- Hierarchy Viewer 112
- Hintergrundbilder 111
- Hochpass 308

## I

- Icon-Menü 211
- ID 32
- IDE 3
- ImageButton 116
  - contentDescription 116

- onClick() 116
- src 116
- ImageView 116
  - contentDescription 116
  - scaleType 116
  - setImageBitmap() 292
  - setImageResource() 292
  - src 116
- import 28
- Importieren
  - Klassen 28
  - Projekte 385
- Innenabstand (Padding) 90
- insert() (SQLiteDatabase) 321
- Installation
  - Android Studio 5
  - JDK (Java) 4
- Intent (Klasse) 248
- Intents 51, 247
  - Action 248
  - Broadcast Intents 52, 257
  - Bundle-Daten 251, 253
  - Category 248
  - Component 248
  - Data 248
  - Daten auslesen 253
  - empfangen 253
  - erzeugen 251
  - explizite 249
  - Extras 248
  - implizite 249
  - Intent-Filter 249
  - senden 252
  - Start-Activity 250
  - zusätzliche Daten mitgeben 251
- interner Speicher 260
- isChecked() (CheckBox) 115
- isProviderEnabled()
  - (LocationManager) 332

## J

- jarsigner 376
- Java
  - JDK 4
  - JRE 4
- Java-Tutorium 409
- JDK (Java) 4
  - Installation 4
- JRE (Java) 4

## K

- Kamera 295
- keystore 376
- KillableAfter-Flag 182
- Klassen

- importieren 28
- Namen 29
- Klickereignisse 158
- Kontextmenüs 211, 217
- Koordinaten, Grafik 199

## L

- Lagesensor 309
- Launcher-Icon ändern 48
- Layout 85
  - Design-Ansicht 92
  - GridLayout 105
  - LinearLayout 100
  - RelativeLayout 101
  - TableLayout 104
- Layoutdateien
  - selbst definierte View-Klassen 193
  - XML-Code 88
- Layout-Designer 88, 92
- Layoutparameter, allg. 96
  - layout\_height 97
  - layout\_marginBottom 99
  - layout\_marginLeft 99
  - layout\_marginRight 99
  - layout\_marginTop 99
  - layout\_width 97
- Layouts 31, 146
  - Attribute 89
  - Designrichtlinien 118
  - Größenangaben 98
  - Hierarchie 94
  - Hoch- und Querformat 126
  - IDs zuweisen 129
  - im Hierarchy Viewer 112
  - laden 128
  - per Code 31
  - per XML 31
  - Portrait/Landscape 126
  - setContentView() 32
  - Stile 148
  - XML-Code 88
  - XML-Dateien 32
- Layout-Views 53, 95
  - FrameLayout 108
  - GridView 107
- Lebenszyklus, App 179
- LIFO-Prinzip 180
- LinearGradient 209
- LinearLayout 100
  - gravity 100
  - layout\_gravity 101
  - layout\_weight 101
  - orientation 100
- Listener-Interfaces 158, 161f.
- Listenfelder 364
- ListFragment 224, 229

- ListView 325
- load() (SoundPool) 283
- Location
  - convert() 335
  - distanceBetween() 335
  - distanceTo() 335
  - getAccuracy() 340
  - getAltitude() 335
  - getBearing() 335
  - getLatitude() 334
  - getLongitude() 334
  - getSpeed() 335
  - getTime() 334
  - hasAccuracy() 340
- LocationListener 332
  - onLocationChanged() 334
- LocationManager 331
  - isProviderEnabled() 332
  - removeUpdates() 333
  - requestLocationUpdates() 333
- Log 184
- Logging 184, 398
- Lösungen
  - zu den Übungen 411

## M

- makeText() (Toast) 244
- Manifestdatei 58
  - Activities eintragen 255
  - Berechtigungen (Permissions) 286
- Margin (Außenabstand) 99, 119
- Market Place 373
- match\_parent 97
- Material zum Buch 409
- MediaController 293
- MediaPlayer 284
  - Audiodateien abspielen 285
  - Audiodateien aus dem Internet abspielen 285
  - Audioressourcen abspielen 284
  - create() 284
  - Endlosschleife 290
  - pause() 285
  - prepare() 288
  - release() 289
  - setDataSource() 288
  - setLooping() 290
  - start() 285
  - stop() 285
  - Systemressourcen freigeben 289
  - wiederverwenden 287
- MediaStore 296
- Mehrsprachigkeit 144, 367
- Menüeinträge ActionBar 215
- MenuInflater 216
- MenuItem 221
  - getItemId() 221

- getMenuInfo() 222
- MenuItem.OnMenuItemClickListener 222
- Menüs 211
  - Action-Menü 212
  - Ereignisbehandlung 221
  - Kontextmenü 211, 217
  - MenüInflater 216
  - Optionen-Menü 211, 216
  - Popup-Menü 212, 219
  - Ressourcen 147, 213
  - Submenüs 211
  - Untermenüs 211, 220
- Methoden
  - Callback 237
  - überschreiben 186
- Minimum SDK-Version 21
- MotionEvent 169
  - ACTION\_DOWN 169
  - ACTION\_MOVE 169
  - ACTION\_POINTER\_DOWN 172
  - ACTION\_POINTER\_UP 172
  - ACTION\_UP 169
  - getAction() 169
  - getX() 171
  - getY() 171
- moveToFirst() (Cursor) 323
- Multimedia 281
  - Audiodateien 284
  - Bilder 292
  - Fotos 295
  - Kamera 295
  - Ressourcen 147
  - Soundeffekte 282
  - Video 293
  - Videos 295
- Multi-Touch 172

## O

- onClick() (ImageButton) 116
- OnClickListener 158, 161
  - onClick() 158, 161
- onClick() (OnClickListener) 158, 161
- onClick() (RadioButton) 117
- onClick() (ToggleButton) 118
- onClose() (SQLiteOpenHelper) 319
- OnCompletionListener 286
  - onCompletion() 286
- onContextItemSelected() (Activity) 221
- onCreate() (Activity) 30
- onCreateContextMenu() (Activity) 218
- onCreateDialog() (Activity) 234
- onCreateDialog() (DialogFragment) 234

- onCreateOptionsMenu() (Activity) 215f.
- onCreate() (SQLiteOpenHelper) 317
- onCreateView() (DialogFragment) 234
- OnDragListener 161
  - onDrag() 161
- onDraw() (View) 191, 195
- OnFocusChangeListener 162
  - onFocusChange() 162
- OnItemClickListener 362
  - onItemClick() 362
- OnItemSelectedListener 366
  - onItemSelected() 366
  - onNothingSelected() 366
- onKeyDown() (View) 207
- OnKeyListener 162, 298
  - onKeyDown() 162
- OnLoadCompleteListener 283
  - onLoadComplete() 283
- onLocationChanged() (LocationListener) 334
- OnLongClickListener 162
  - onLongClick() 162
- onOptionsItemSelected() (Activity) 221
- onSensorChanged() (SensorEventListener) 302f.
- onTouchEvent() (View) 172
- OnTouchListener 162, 168
  - onTouch() 162, 168
- onTouch() (Switch) 117
- onUpgrade() (SQLiteOpenHelper) 319
- openFileInput() (Activity) 261
- openFileOutput() (Activity) 261
- Optionen-Menü 211, 216
- Overflow-Menü 215

## P

- package 28
- PackageManager 257
- Padding (Innenabstand) 90, 119
- Paint 192
  - setAlpha() 209
  - setColor() 196
  - setStrokeWidth() 196
  - setStyle() 199
- Pakete 20, 28, 56
- parse() (Uri) 286
- Path 209
- pause() (MediaPlayer) 285
- Permission
  - READ\_EXTERNAL\_STORAGE 265
  - WRITE\_EXTERNAL\_STORAGE 265
- Permissions 265
- Plattformen (Android) 10

- play() (SoundPool) 284
- Pointer 172
- PointerCoords 172
- Popup-Menü 212, 219
- postInvalidate() (View) 271
- Preferences 259
- prepare() (MediaPlayer) 288
- ProgressBar 116
  - max 116
  - progress 116
  - style 116
- ProgressDialog 240
- Projekte
  - anlegen 18
  - auf der Festplatte 25
  - Dateien 55
  - Grundgerüst 26
  - Projektverzeichnis 25
  - Wizards 18
- Properties 94

## Q

- query() (SQLiteDatabase) 322
- Quiz-App 273

## R

- RadioButton 117
  - checked 117
  - onClick() 117
  - text 117
- RadioGroup 117
  - checkedButton 117
  - orientation 117
- random() (Math) 330
- Reaktions-App 267
- RectF 199
- Referenz, der Android-API 12
- registerForContextMenu() (Activity) 218
- register() (Sensor) 302
- RelativeLayout 101
  - layout\_above 103
  - layout\_align... 103
  - layout\_below 103
  - layout\_center... 103
  - layout\_toLeftOf 103
  - layout\_toRightOf 103
- release() (MediaPlayer) 289
- removeUpdates() (LocationManager) 333
- requestLocationUpdates() (LocationManager) 333
- Ressourcen 34, 133
  - als Objekte laden 139

- alternative Ressourcenversionen 152
  - an Attribute zuweisen 137
  - anlegen 134
  - an View-Eigenschaften zuweisen 136
  - Audiodateien 281
  - Bilder 145
  - Dateien 264
  - Dateinamen 135
  - Datenbanken 319
  - entfernen 140
  - Farben 141
  - Format 134
  - Größenangaben 140
  - im Code 139
  - Layouts 146
  - Lokalisierung 144
  - Mehrsprachigkeit 367
  - Menüs 147, 213
  - Multimedia 147
  - Rohdaten 147
  - Speicherort 135
  - Stile 148
  - Strings 142
  - verwenden 136
  - Videodateien 281
  - XML-Drawable 145
  - Ressourcendateien 135
  - R.java 32, 56, 136, 138
  - R-Klasse 32
  - Rohdaten 147
  - Root-Activity 180
  - Run-Konfigurationen 385
- S**
- scale() (Canvas) 198
  - Schlüssel 375
  - SDK
    - für Android 3, 9
    - Version 21
  - SD-Karte
    - Emulator 390
    - Test auf Existenz 266
    - Zugriff 264
  - sendMessageDelayed() (Handler) 356
  - sendMessage() (Handler) 355
  - Sensor 300, 302
    - register() 302
    - Typen-Konstanten 299f.
  - Sensoren 299
    - bei Sensor registrieren 302
    - Beschleunigungssensor 305
    - Daten auslesen 303
    - Filter 306
    - Lagesensor 309
    - Sensortypen 299f.
    - verfügbare Sensoren 300
    - Werte 304
  - SensorEvent 300, 304
  - SensorEventListener 300, 302
    - onSensorChanged() 302f.
  - SensorManager 300f.
    - getDefaultSensor() 301
    - getSensorList() 301
  - Services 53
  - setAlpha() (Paint) 209
  - setColor() (Paint) 196
  - setContentView() (Activity) 30, 32, 128
  - setDataSource() (MediaPlayer) 288
  - setGravity() (Toast) 244
  - setImageBitmap() (ImageView) 292
  - setImageResource() (ImageView) 292
  - setLooping() (MediaPlayer) 290
  - setStrokeWidth() (Paint) 196
  - setStyle() (Paint) 199
  - SharedPreferences 259
  - show() (Toast) 244
  - Signieren 375
  - SimpleCursorAdapter 325
  - Somigliana 306
  - Sound 281
    - Audiodateien 284
    - MediaPlayer 284
    - Soundeffekte 282
    - SoundPool 282
    - Töne 290
  - SoundPool 282
    - load() 283
    - play() 284
  - Spinner 117, 364
    - Ereignisbehandlung 366
    - konfigurieren 364
    - mit Daten füllen 365
    - onItemSelected 117
    - prompt 117
  - Sprites 203
  - SQL 316
  - SQLiteDatabase 317
    - close() 319
    - delete() 324
    - execSql() 318
    - insert() 321
    - query() 322
    - update() 324
  - SQLiteOpenHelper 316
    - getReadableDatabase() 317
    - getWritableDatabase() 317
    - onClose() 319
    - onCreate() 317
    - onUpgrade() 319
  - Start-Activity 58
  - startActivity() (Activity) 252
  - startActivityForResult() (Activity) 268
  - start() (MediaPlayer) 285
  - startTone() (ToneGenerator) 290
  - StatFs 267
  - Stile 148
    - an Activities zuweisen 151
    - an Views zuweisen 148
    - definieren 148
    - parent-Attribut 151
    - Themes 151
    - Vererbung 150
  - stop() (MediaPlayer) 285
  - stopTone() (ToneGenerator) 291
  - Strings 34, 142
    - in Ressourcen verwandeln 34
  - strings.xml 34
  - style-Attribut 148
  - Switch 117
    - checked 117
    - onTouch() 117
    - text 117
    - textOff 117
    - textOn 117
  - Syntaxhervorhebung 64
- T**
- TableLayout 104
  - TableRow 104
  - Target-SDK
    - eines Projekts 21
    - nachträglich ändern 47
  - Task 180
  - Tastaturereignisse 207
  - Testen
    - auf Smartphone 45
    - im Emulator 38
  - TextView 118
    - text 118
    - textSize 118
    - textStyle 118
    - typeface 118
  - Themes 151
  - Threads 241, 270
  - TicTacToe-App 341
  - Tiefpass 307
  - TimePickerDialog 237f.
  - Timer 83
  - TimerTask 83
  - Tippereignisse 168
  - Toast 244
    - makeText() 244
    - setGravity() 244
    - show() 244
  - Toasts 158, 244
  - ToggleButton 118

- checked 118
- onClick() 118
- textOff 118
- textOn 118
- ToneGenerator 290
- startTone() 290
- stopTone() 291
- Tools

- jarsigner 376
- keystore 376
- translate() (Canvas) 198

## U

- UFO-App 203
- UI *siehe* Benutzeroberflächen
- UI-Elemente 114
- Untermenüs 211, 220
- update() (SQLiteDatabase) 324
- Uri 285
  - encode() 286
  - parse() 286
- USB-Debugging einstellen 45

## V

- Veröffentlichung 373
  - Screenshots der App 397
- Vibrator 353
  - vibrate() 353
- Vibrieren 353
- Video 281

- MediaPlayer 293
- Ressourcen 281
- Videos 295
- VideoView 293
- View 85
  - eigene View-Klassen erzeugen 193
  - eigene View-Klassen in XML 193
  - onDraw() 191
  - onKeyDown() 207
  - onTouchEvent() 172
  - postInvalidate() 271
- ViewGroup 85
  - addView() 195
  - Layoutparameter 96
  - Layoutregeln 96
- ViewGroups 53, 99
- Views 52
  - Attribute 89
  - Drehung 90
  - Eigenschaften 94
  - Fokussierbarkeit 121
  - Hierarchie 94
  - Hintergrund 90, 109
  - Hintergrundbild 111
  - Hintergrundfarbe 109
  - ID 90
  - Innenabstand (Padding) 90
  - Kontextmenüs 217
  - Layout-Views 53, 95
  - mit ID verbinden 129
  - Sichtbarkeit 90
  - Transparenz 90

- ViewGroups (Container) 53, 99
- Widgets 53, 114
- Zeichenflächen 53
- zeichnen 191
- Zugriff in Code 130

## W

- Website, zu Buch 14
- WebView 118
- WebViewFragment 224
- Widgets 53, 114
- Wischereignisse 170
- Wizards 18
- wrap\_content 97

## X

- XML-Drawable 145
- xml-Layouts 32

## Z

- Zeichenflächen 53
- Zeichnen 196
- Zeit, Auswahl über Dialog 238
- Zertifikat, digitales 375
- Zoomen 174
- Zufallsgenerator 271
- Zugriffsrechte 265
- Zurück-Taste 120